状態遷移を利用した高速ポリライン・クリップ方式†

向井 信彦††

2次元グラフィックスのクリップ処理において，1つ前の点が存在する領域の情報を活用することにより，高速なポリラインのクリップ・アルゴリズムを開発した．従来のクリップ方式では，線分を構成する両端点が与えられた場合のアルゴリズムしか提案されておらず，効率の良いポリラインのクリップ処理アルゴリズムは明確ではなかった．本論文では，クリップ処理の際に1つ前の点が存在する領域の情報を活用し状態を遷移させることにより，高速にポリラインをクリップするアルゴリズムを提案する．近年，高速プロセサの発展により演算能力は非常に向上している．しかしながら，これらの高速プロセサを用いた場合の問題点の1つとして，分岐命令によるパイプラインの性能劣化がある．そこで，本アルゴリズムでは分岐命令を極力少なくするとともに，クリップ・インおよびクリップ・アウトの両場合ともに高速処理を行えるようにした．定量的な評価として，2点を基にクリップ処理を行う従来のアルゴリズム，および Cohen-Sutherland 法を用いて1つ前の4ビット・コードを保存し1点処理を行うように変更したアルゴリズムと比較した．比較の結果，両場合に比べて本方式ではクリップ処理に必要な比較および分岐命令数が大幅に減少しており，高速なクリップ処理の実現性を確認できた．

1. はじめに

近年におけるグラフィックス利用者の急増と，ハードウェア技術の著しい進歩により，従来に比べて非常に安価で高性能なグラフィック・ワークステーションが出現してきた．また，グラフィックスの標準化も定着し，GKS, PHIGS 等が広く利用されるようになってきた[1),2)]．

これらグラフィック・ワークステーションの用途は様々であるが，主には CAD/CAM 分野である．これらの分野で利用する際，特に重要となるのがクリップ処理である．CAD/CAM 分野における製図では，数ミクロン単位のものから数十キロ・メートルのものまでを扱う可能性があり，これらのものを表示するためには，拡大あるいは縮小して表示することになる．元図形を拡大して表示する場合，ほとんどのプリミティブはクリップ・アウトされ，ごく一部のプリミティブのみが表示される．逆に，縮小して表示する場合は，全プリミティブが表示される．つまり，クリップ・インとクリップ・アウトの両場合とも，高速に判断する必要がある．

従来におけるクリップ処理方法の代表例としては，Cohen-Sutherland の方法がある[3),4)]．この方式は，クリップ枠を構成する4直線と描画対象となる線分から，線分の両端点における4ビットのコードを作成して，クリップ判断を行うものである．この方式における4ビット・コードの作成は極めて容易であるが，常に両端点の4ビット・コードを求めなければならないという欠点がある．つまり，最小2回の比較でクリップ・アウトの判定を行える場合にも，両端点（2点）の4ビット・コードを求めるために合計8回の比較が必要となる．

この Cohen-Sutherland (CS) 手法を応用して，クリップ枠と線分との交点を求めるために，常に線分の中点を分割点に選ぶ方法も発表されている[5)]．この中点を分割点に選ぶ手法は，加算器とシフタのみで実現でき，非常にコンパクトでかつハードウェア化に適した高速な手法であるが，繰返し回数が多くなるのと最大の繰返し数を考慮した十分な演算精度を持たない限り，クリップされた線分の端点精度が悪くなるという欠点を持っている．

また，Liang と Barsky によって，非常に効率的なパラメータを用いた手法が発表された．この Liang-Barsky (LB) 手法を用いると，実クリップ（クリップ判定ではなく実際のクリップ処理）を高速に行うことはできるが，パラメータ計算に4回の除算を伴う．このため，クリップ・インあるいはクリップ・アウトという単なるクリップ判定のみの場合，逆に遅くなる可能性がある[6)]．

これに対して，クリップ対象となる線分の空間的位置を考慮して，最小の演算回数でクリップ処理を行う方式が，2人の Nicholl と Lee によって発表された[7)]．この手法では，最も簡単なクリップ・アウトの

† Fast Polyline Clipping Algorithm Using State Transition by NOBUHIKO MUKAI (Engineering Computer Architecture Group, Engineering Computer Systems Department, Information Systems and Electronics Development Laboratory, Mitsubishi Electric Corporation).

†† 三菱電機(株)情報電子研究所エンジニアリング・システム開発部アーキテクチャ・グループ

場合，2回の比較で判定することができる．また，実クリップも最小限の演算数で処理できる（特に，除算回数は必要最小限である）．ところが，この Nicholl-Lee-Nicholl (NLN) の方法では，処理速度の評価として算術演算の基本である四則演算および比較の回数のみを考察の対象としており，比較後の分岐命令を考慮に入れていない．

一方ここ数年来，非常に高速なプロセサが開発されている[8)~10)]が，基本的にはパイプライン処理による高速化である．この場合，一連の命令列の途中で分岐が発生すると，パイプラインに乱れが生じ，高速処理が実現されなくなる．つまり文献7)では，基本的な演算性能は正しく評価しているが，分岐命令によるパイプラインの性能劣化という問題に対しては，十分な性能評価がなされていないことになる．

従来のクリップ方式では，線分を構成する両端点（2点）が与えられた場合のアルゴリズムしか提案されておらず，GKS や PHIGS で用いられている「ポリライン」に適した効率のよいアルゴリズムは明確ではなかった．そこで本論文では，クリップ処理の際に1つ前の点が存在する領域の情報を活用し状態を遷移させることにより，高速にポリラインをクリップするアルゴリズムを提案する．

2. アルゴリズムの概要

1章でも述べたように，Nicholl-Lee-Nicholl (NLN) 法は，クリップ対象となる線分の空間的位置を考慮し，最小の演算回数でクリップ処理を行う方式であり，従来技術の中では最も高速であると思われる．したがって，本アルゴリズムの基本は，NLN 法とする．ただ，NLN 法は線分を構成する両端点（2点）が与えられた場合の説明しかなく，ポリラインのような連続線分に適したクリップ方式については言及していない．

本論文では NLN 法を核として，一線分のクリップ処理後に線分の後点（つまり，線分をポリラインの一部と考えたとき，順番上，後に来る点で，単一線分の場合の終点に相当する点である．また，単一線分の場合の始点に相当する点を前点と呼ぶことにする）が存在する領域を明確にし，この情報を用いてクリップ処理の状態を遷移させることにより，高速にポリラインをクリップするアルゴリズムを提案する．線分の後点が存在する領域情報を活用することにより，アルゴリズムをさらに最適化することができる．例えば，NLN 法では図形の対象性を考慮してアルゴリズムの簡潔な記述を追求するため，メイン・ルーチンにおける領域の分割が次の3つに限定されている．

$$x<xL \quad \text{or} \quad xR<x \quad \text{or} \quad xL\leq x\leq xR$$

ただし，(x, y) は線分を構成する点の座標であり，xL, xR, yB および yT をクリップ枠を構成する線分の x あるいは y 座標とする．

ところが，図形の対象性を考えれば，当然次のような領域分割も可能である．

$$y<yB \quad \text{or} \quad yT<y \quad \text{or} \quad yB\leq y\leq yT$$

本論文では，これら図形の対象性も考慮してクリップ処理の状態をうまく遷移させることにより，最小の命令数でポリラインのクリップを行うアルゴリズムを提案する．

また，NLN 法は線分のクリップ・アウトを優先的に扱っているため，クリップ・アウトに関しては最小命令数で高速に処理を行えるが，クリップ・インはクリップ・アウト判断の結果として最後まで残ったものとして扱われる．このため，クリップ・インは非常に

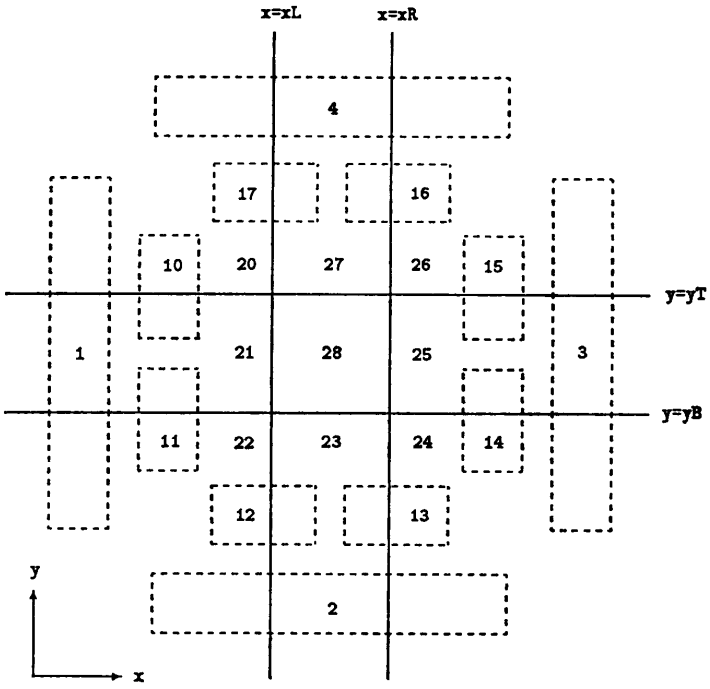


図 1 クリップ処理において線分の端点が存在する領域
Fig. 1 Regions in which the point of a line exits in clipping processes.

region1	: $x < xL$		
region2	:		$y < yB$
region3	: $xR < x$		
region4	:		$yT < y$
region10	: $x < xL$	and	$yB < y$
region11	: $x < xL$	and	$y < yT$
region12	: $x < xR$	and	$y < yB$
region13	: $xL < x$	and	$y < yB$
region14	: $xR < x$	and	$y < yT$
region15	: $xR < x$	and	$yB < y$
region16	: $xL < x$	and	$yT < y$
region17	: $x < xR$	and	$yT < y$
region20	: $x < xL$	and	$yT < y$
region21	: $x < xL$	and	$yB \leq y \leq yT$
region22	: $x < xL$	and	$y < yB$
region23	: $xL \leq x \leq xR$	and	$y < yB$
region24	: $xR < x$	and	$y < yB$
region25	: $xR < x$	and	$yB \leq y \leq yT$
region26	: $xR < x$	and	$yT < y$
region27	: $xL \leq x \leq xR$	and	$yT < y$
region28	: $xL \leq x \leq xR$	and	$yB \leq y \leq yT$

図 2 線分の端点が存在する領域の定義
Fig. 2 Definition of regions in which the point of a line exits.

遅くなる．クリップ処理の状態をうまく遷移させることにより，場合に応じてクリップ・イン処理をクリップ・アウト処理よりも優先させて行うことは可能である．しかしながら，NLN 法で用いられているように，点の座標とクリップ枠とを比較し結果に応じて分岐する方法では，クリップ・イン処理の場合，分岐命令が非常に多くなる．このことは，高速プロセサのパイプライン処理を乱す原因となり，性能の劣化につながる．一方，Cohen-Sutherland (CS) 法のように4ビット・コードを作成すれば，分岐命令の数は大幅に減少する．

3. クリップ・アルゴリズム

3.1 状態の種類

2次元グラフィックスにおけるクリップ枠を構成する4直線を $x=xL$, $x=xR$, $y=yB$ および $y=yT$ とする．この4直線によって9つの小領域が生じるが，クリップ対象となる線分の端点が存在する可能性のある領域は，**図1**および**図2**に示す 21 種類の領域である（ここで，例えば領域1とは領域 20, 領域 21 および領域 22 という3つを包含する領域である）．本論文では，線分を構成している両端点のうち1点（後点）がこれらの領域のいずれに存在するかが判明した時点で，「クリップ処理は状態 X にある」ということにする．つまり，1線分のクリップ処理が終了した時点で線分を構成している両端点のうち後点の存在する領域が判明するが，例えば，後点が図1に示す1の領域に存在していることが判明すれば，「クリップ処理は状態1にある」という．線分の端点が存在する 21 の領域は言い替えれば，「2次元クリップ処理で可能性のある状態は 21 種類である」ということになる．

3.2 第1点の領域情報

本アルゴリズムは，ポリラインを高速にクリップする目的で作成したものである．したがって，基本的には連続した点列のうち第2点以降の処理が中心となるが，第1点の領域情報を求めておかなければ，第2点以降のクリップ処理を行うことはできない．ここでは，第1点の領域情報を求める手順について説明する．

クリップ枠を構成する4直線を図1と同様に $x=xL$, $x=xR$, $y=yB$ および $y=yT$ とし，第1点の座標を $(x0, y0)$ とすると，第1点の領域情報を求めるアルゴリズムは**図3**のようになる．

3.3 状態が1桁の場合

状態が1桁の場合は，1, 2, 3 および4の4種類である．図形の対象性を考慮すれば，これらのうちの1つを例に取って説明すれば十分である．ここでは，状態1を例に取って説明する．状態1の場合は，1つ前のクリップ処理の結果，線分を構成している両端点のうち後点の x 座標が xL より小さいことを意味する．言い替えると，次のクリップ処理において，線分を構成する両端点のうち前点の x 座標が xL より小さいことと同じである．このことから，次のクリップ処理における線分を構成している後点の x 座標も xL より小さいと予想できるため，まず最初に，後点の x 座標と xL との比較を行う．この結果，後点の x 座標が xL よりも小さい場合は，クリップ・アウトと判定される．

```
first_state()
 {
  if x0<xL then goto state_1();
  else if x0>xR then goto state_3();
       else if y0<yB then goto state_23();
            else if y0>yT then goto state_27();
                 else then goto state_28();
 }
```

図 3 第1点の領域情報を求めるアルゴリズム
Fig. 3 Algorithm which decides the state information of the first point.

```
state_1()
 {
  if x2<xL then クリップ・アウト,goto state_1();
  else if y1<yB then goto label_22 in state_22();
       else if y1>yT then goto label_20 in state_20();
            else then goto label_21 in state_21();
 }
```

図 4 状態が1の場合
Fig. 4 In the case of state 1.

```
state_10()
 {
  if x2<xL then クリップ・アウト,goto state_1();
  else if y1>yT then goto label_20 in state_20();
       else then goto label_21 in state_21();
 }
```

図 5 状態が10の場合
Fig. 5 In the case of state 10.

後点の x 座標が xL より小さくない場合には，前点の y 座標と yB あるいは yT との比較を行い，結果に応じて状態を遷移させる．例えば，前点の y 座標が yB よりも小さい場合には前点の x 座標は xL よりも小さく，かつ前点の y 座標は yB よりも小さいことが判明するので状態を状態 22 に遷移させる．また前点の y 座標が，yT よりも大きい場合には前点の x 座標は xL よりも小さく，かつ前点の y 座標は yT よりも大きいことが判明するので状態を状態 20 に遷移させる．それ以外の場合は前点の x 座標は xL よりも小さく前点の y 座標は yB 以上，yT 以下であると判明するので状態を状態 21 に遷移させる．クリップ対象となる線分を構成する始点（ポリラインの一部と考えるときは前点）の座標を $(x1, y1)$，終点（ポリラインの一部と考えるときは後点）の座標を $(x2, y2)$ とすれば，状態が1の場合のアルゴリズムは図4のようになる．

状態が2，3および4の場合には，図形の対象性を考慮して x, y, xL, xR, yB および yT を適当に入れ換えれば良い．

3.4 状態が 10 番台の場合

状態が 10 番台の場合は，10 から 17 まで8種類ある．この場合についても同様に，図形の対象性を考慮すれば，状態 10 についての説明で十分である．アルゴリズムを図5に示す．

3.5 状態が 20 番台の場合

状態が 20 番台の場合は，20 から 28 までの9種類ある．この場合についても同様に，図形の対象性から状態 20，21 および 28 の3種類について説明すれば十分である．この場合，基本的なクリップ処理方式はNLN法と同じである．状態 20 および状態 21 のアルゴリズムを，図6および図7に示す．

```
        state_20()
         {
          if x2<xL then クリップ・アウト,goto state_1();
label_20: else if y2>yT then クリップ・アウト,goto state_16();
          else then {
            if (xL-x1)*(y2-y1) < (yT-y1)*(x2-x1) then
             {
              if y2<yB then
               {
                if (xL-x1)*(y2-y1) < (yB-y1)*(x2-x1)
                  then クリップ・アウト,goto state_13();
                else then {
                     x=xLでクリップ;
                     if x2>xR then {
                        if (xR-x1)*(y2-y1) > (yB-y1)*(x2-x1)
                           then x=xRでクリップ,goto state_24();
                        else then y=yBでクリップ,goto state_24(); }
                     else then y=yBでクリップ,goto state_23(); }
               }
              else then {
                   x=xLでクリップ;
                   if x2>xR then x=xRでクリップ,goto state_25();
                   else then goto state_28(); }
             }
            else then {
              if x2>xR then
               {
                if (xR-x1)*(y2-y1) > (yT-y1)*(x2-x1)
                  then クリップ・アウト,goto state_14();
                else then {
                     y=yTでクリップ;
                     if y2<yB then {
                       if (xR-x1)*(y2-y1) < (yB-y1)*(x2-x1)
                          then y=yBでクリップ,goto state_24();
                       else then x=xRでクリップ,goto state_24(); }
                     else then x=xRでクリップ,goto state_25(); }
               }
              else then {
                   y=yTでクリップ;
                   if y2<yB then y=yBでクリップ,goto state_23();
                   else then goto state_28(); }
             }
          }
        }
```

図 6 状態が 20 の場合
Fig. 6 In the case of state 20.

```
        state_21()
         {
          if x2<xL then クリップ・アウト,goto state_1();
          else then
              {
label_21:      if y2<yB then
                 {
                  if (xL-x1)*(y2-y1) < (yB-y1)*(x2-x1)
                     then クリップ・アウト,goto state_13();
                  else then {
                       x=xLでクリップ;
                       if x2>xR then {
                          if (xR-x1)*(y2-y1) > (yB-y1)(x2-x1)
                             then x=xRでクリップ,goto state_24();
                          else then y=yBでクリップ,goto state_24(); }
                       else then y=yBでクリップ,goto state_23(); }
                 }
                else if y2>yT then
                 {
                  if (xL-x1)*(y2-y1) > (yT-y1)*(x2-x1)
                     then クリップ・アウト,goto state_16();
                  else then {
                       x=xLでクリップ;
                       if x2>xR then {
                          if (xR-x1)(y2-y1) < (yT-y1)(x2-x1)
                             then x=xRでクリップ,goto state_26();
                          else then y=yTでクリップ,goto state_26(); }
                       else then y=yTでクリップ,goto state_27(); }
                 }
                else then {
                   x=xLでクリップ;
                   if xR<x2 then x=xRでクリップ,goto state_25();
                   else then goto state_28(); }
              }
         }
```

図 7 状態が 21 の場合
Fig. 7 In the case of state 21.

3.6 状態が 28 の場合

2章でも述べたように，本アルゴリズムでは後点が存在する領域情報を活用して状態をうまく遷移させることにより，最適なクリップ処理を行うことができる．つまり，1つ前のクリップ処理において線分を構成する両端点のうち後点の座標がクリップ枠内にある場合には，次のクリップ処理においても線分を構成している両端点のうち後点の座標はクリップ枠内に存在すると予想されるため，クリップ・インを優先させて処理が行える．

この結果，クリップ・インの処理は高速になる．ところが，NLN 法で用いられているように点の座標とクリップ枠とを比較し，その結果に応じて分岐を行っていると分岐命令が多くなり，パイプライン性能の劣化を引き起こす原因になる．

そこで，本アルゴリズムでは状態が 28 の場合に限り，分岐命令数を少なくしてクリップ・インを高速に処理するために，Cohen-Sutherland (CS) 法の4ビット・コードを用いる．つまり，線分を構成している両端点の座標をクリップ枠と比較するのではなく，両端点の4ビット・コードを作成し，このコードを基にクリップ処理を行う．調べようとする点の座標を (x, y) とすると，4ビット・コードは次式における計算の符号を取り出すことによって得られる．

$$(yT-y, y-yB, xR-x, x-xL)$$

図 8 に，各領域における4ビット・コードの2進数値を，また括弧の中に 10 進数値を示す．また，この4ビット・コードを利用した，状態 28 のクリップ処理アルゴリズムを**図 9** に示す．

4. 性能評価

本論文で提案したアルゴリズムを NLN 法と比較し，性能を評価した．ただ，NLN 法は線分を構成する2端点が与えられた場合のアルゴリズムであるため，参考として，CS 法の1つ前のクリップ処理における4ビット・コードを保存し，次のクリップ処理においては1点の4ビット・コードを作成すればよい方法と比較した．ここでは簡単化のために，2点処理の基本アルゴリズムを NLN (Nicholl-Lee-Nicholl) 法，1つ前の4ビット・コードを保存するように変更した Cohen-Sutherland 法を UCS (Updated Cohen-Sutherland) 法，本論文で提案する方法を ST (State Transition) 法と呼ぶことにする．

評価の尺度としては，文献 7) で採用している比較および四則演算に分岐数を加えた．また CS 法に関しては，原本[3]よりも改良版[4]の方が，分岐命令を少なくして実現できるため，こちらのアルゴリズムを基に改良を加えた．文献 7) では，CS 法におけるクリップ・インの条件判定において，2つの4ビット・コードと0との比較には2回の比較命令が必要であると考えているが，2つのコードの論理和を取って比較すれば，比較回数を1回に減少させることができる．したがって，本論文での比較回数もこの考えを採用して算出している．

結果を，**図 10～図 12** に示す．図 10 は始点が領域 20, 22, 24 および 26 にある場合の各処理ステップを，図形およびアルゴリズムの対象性を考慮して平均したものである．また，図 11 は始点が領域 21, 23, 25 および 27 にある場合について，同様に平均を取っ

たものである．両場合とも，＊印を付した領域は場合に応じて一部の線分がクリップ枠内に入る場合と完全にクリップ・アウトされる2つの場合が存在するが，

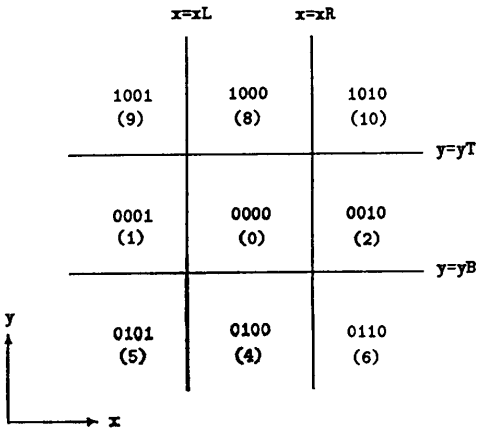


図 8 CS 法の4ビット・コード
Fig. 8 4 bit code used in CS algorithm.

ここでは前者の場合についての処理ステップを表している．ST 法の比較および分岐以外の命令については，NLN 法と同じである．

また図では，状態が 20 番台のものを示しているが，状態が1桁あるいは 10 番台のものについても，ほぼ同様である．例えば，始点の状態が1の場合を例に取って説明する．図 10 において，終点が $x=xL$ よりも左側にある3つの場合は，始点の状態が1の場合も始点の状態が 20 の場合（つまり図 10）と同一である．図4のアルゴリズムに従い始点が状態 20 にあると判断された場合，図 10 において終点が $x=xL$ より右側にある 6 つの場合は，比較が2回（$y1<yB$ および $y1>yT$），分岐が1回 (goto label_20) 増加するだけである．状態が 10 の場合も同様であり，図5のアルゴリズムに従い始点が状態 20 にあると判断された場合，$x=xL$ の左側にある3つの場合は同一で，右側にある6つの場合は比較が1回（$y1>yT$），分岐

```
state_28()
 {
  (yT-y2,y2-yB,xR-x2,x2-xL)の符号から4ビット・コード生成;

  in case (4bit_code=0)
         {
           goto state_28();
         }
  in case (4bit_code=1)
         {
           x=xLでクリップ,goto state_21();
         }
  in case (4bit_code=2)
         {
           x=xRでクリップ,goto state_25();
         }
  in case (4bit_code=4)
         {
           y=yBでクリップ,goto state_23();
         }
  in case (4bit_code=5)
         {
          if (xL-x1)(y2-y1) < (yB-y1)(x2-x1) then x=xLでクリップ;
          else then y=yBでクリップ;
          goto state_22();
         }
  in case (4bit_code=6)
         {
          if (xR-x1)(y2-y1) > (yB-y1)(x2-x1) then x=xRでクリップ;
          else then y=yBでクリップ;
          goto state_24();
         }
  in case (4bit_code=8)
         {
           y=yTでクリップ,goto state_27();
         }
  in case (4bit_code=9)
         {
          if (xL-x1)(y2-y1) > (yT-y1)(x2-x1) then x=xLでクリップ;
          else then y=yTでクリップ;
          goto state_20();
         }
  in case (4bit_code=10)
         {
          if (xR-x1)(y2-y1) < (yT-y1)(x2-x1) then x=xRでクリップ;
          else then y=yTでクリップ;
          goto state_26();
         }
 }
```

図 9 状態が28の場合
Fig. 9 In the case of state 28.

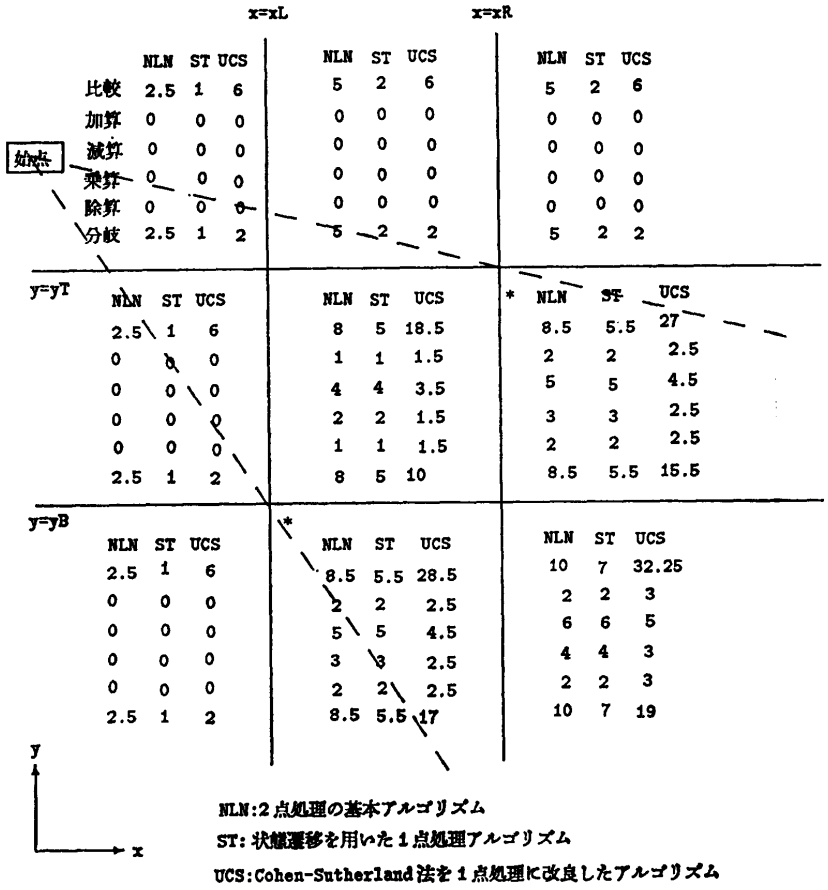


図 10 始点が領域 20 (22, 24, 26) にある場合
Fig. 10 In the case that start point is in the region 20 (22, 24, 26).

が1回 (goto label_20) 増加するだけである.

図からも理解されるように, ST 法は UCS 法あるいは NLN 法に比べて, 比較および分岐の命令数が大幅に減少している. これは, 状態遷移の考えに基づき, 1つ前の点が存在する領域に次の点も存在するであろうという仮定に基づいてアルゴリズムを最適化したためである.

また, 分岐命令数が大幅に減少している点に注目する必要がある. なぜなら, 近年開発されている高速プロセサを用いた場合, 四則演算および比較は1クロックで実行できるが, 分岐命令は1クロックでは実行できない. このため, 様々な手法が用いられている. つまり, 分岐命令が高速化のボトル・ネックとなっており, この分岐命令数を減少させることが高速化につながるわけである.

2章でも説明したように, 本アルゴリズムでは 21 種類の状態を示す領域を考えている. これは, 線分の終点が含まれる可能性のある領域であり, この 21 種類以外には存在しない. これは, クリップ過程を辿れば自ずと得られる結果である. つまり, 線分の端点が存在する領域を, 2本のクリップ枠で分割された9つの領域に振り分けるのでなく, クリップ処理の結果として得られた状態を考え, この状態情報を基に次のクリップ処理を行えば, 余分な演算命令を省き高速化を実現することができるわけである.

5. ま と め

本論文では, 1つ前のクリップ処理において線分を構成する両端点のうち後点が存在する領域の情報を活用することにより, 高速なポリラインのクリップ・アルゴリズムを提案した. 本アルゴリズムは, 状態を遷移させることにより高速化を図ったものであるが, それだけではなく, 分岐命令を極力少なくすることによりパイプラインの性能劣化を抑えることもできた.

性能評価の結果, 2点処理を基本とした従来アルゴリズムおよびクリップ・アルゴリズムの基本である

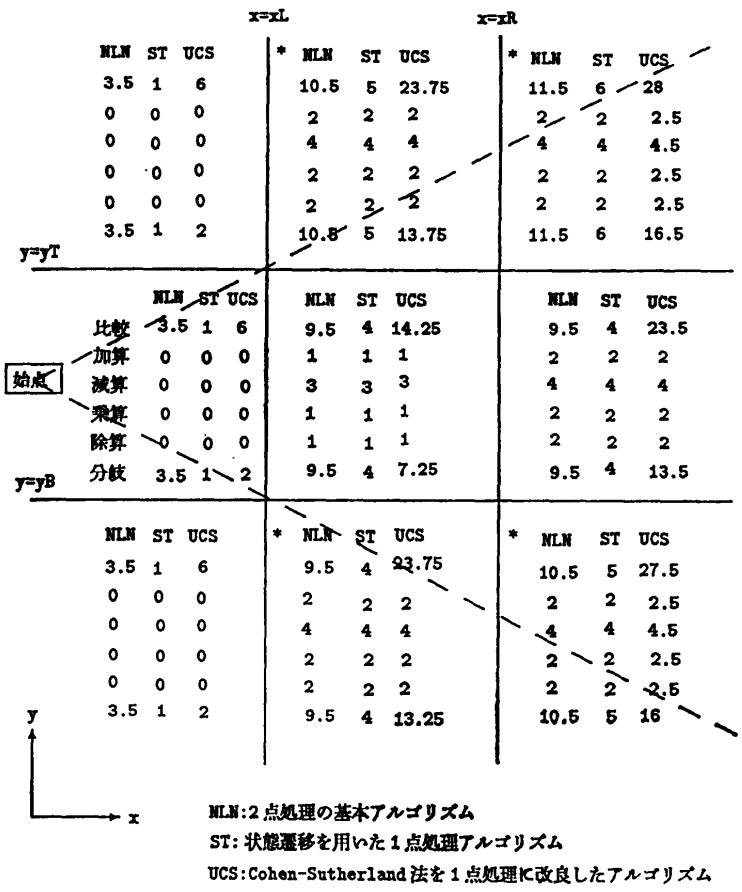


図 11 始点が領域 21 (23, 25, 27) にある場合
Fig. 11 In the case that start point is in the region 21 (23, 25, 27).

Cohen-Sutherland 法を 1 点処理に改良したものに比べて，比較および分岐命令数が大幅に減少しており，このことが性能向上の要因と考えられる．本方式を用いることにより，今後さらに高速な CAD システムが期待される．

謝辞 最後に，本研究の機会を与えてくださいました当社情報電子研究所エンジニアリング・システム開発部渡辺治部長，アーキテクチャ・グループ風間成介グループ・マネージャー，ならびに数々の助言をしていただいたアーキテクチャ・グループの皆様に深く感謝いたします．

参 考 文 献

1) 3次元グラフィックス・ワークステーションを解剖する，日経 CG, No. 9, pp. 10-24 (1987).
2) グラフィックス・ライブラリ：オブジェクト指向と3次元 CG 機能を取り入れて進化，日経 CG, No. 33, pp. 8-19 (1989).
3) Newman, W. M. and Sproull, R. F.: *Principles of Interactive Computer Graphics*, 2nd ed., pp. 63-68, McGraw-Hill, New York (1979).
4) 山口：コンピュータディスプレイによる図形処理工学，pp. 138-145, 日刊工業新聞社 (1981).
5) Sproull, R. F. and Sutherland, I. E.: A Clipping Divider, *Fall Joint Computer Conference*, Vol. 33, pp. 765-775 (1968).
6) Liang, Y. D. and Barsky, B. A.: A New Concept and Method for Line Clipping, *ACM Transactions on Graphics*, Vol. 3, No. 1, pp. 1-22 (1984).
7) Nicholl, T. M. et al.: An Efficient Algorithm for 2-D Line Clipping—Its Development and Analysis, *ACM Comput. Gr.*, Vol. 21, No. 4, pp. 253-262 (1987).
8) 矢野：マイクロプロセサの命令セットアーキテクチャ，情報処理，Vol. 29, No. 12, pp. 1420-1427 (1988).
9) 鹿子木：プロメテウスの火—マイクロプロセッ

x=xL　　x=xR

NLN	ST	UCS
7	5	18
1	1	1.5
4	4	4.5
2	2	1.5
1	1	1.5
7	1	9.5

NLN	ST	UCS
7	4	15
1	1	1
3	3	3
1	1	1
1	1	1
7	1	8

NLN	ST	UCS
8	5	19
1	1	1.5
4	4	4.5
2	2	1.5
1	1	1.5
8	1	10.5

y=yT

NLN	ST	UCS
7	4	13
1	1	1
3	3	3
1	1	1
1	1	1
7	1	6

始点	NLN	ST	UCS
比較	8	4	5
加算	0	0	0
減算	0	0	0
乗算	0	0	0
除算	0	0	0
分岐	8	1	1

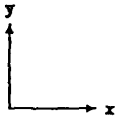
NLN	ST	UCS
8	4	14
1	1	1
3	3	3
1	1	1
1	1	1
8	1	7

y=yB

NLN	ST	UCS
8	5	18
1	1	1.5
4	4	4.5
2	2	1.5
1	1	1.5
8	1	9.5

NLN	ST	UCS
8	4	15
1	1	1
3	3	3
1	1	1
1	1	1
8	1	8

NLN	ST	UCS
9	5	19
1	1	1.5
4	4	4.5
2	2	1.5
1	1	1.5
9	1	10.5



NLN:2 点処理の基本アルゴリズム
ST: 状態遷移を用いた 1 点処理アルゴリズム
UCS:Cohen-Sutherland 法を 1 点処理に改良したアルゴリズム

図 12 始点が領域 28 にある場合
Fig. 12 In the case that start point is in the region 28.

サ㊙, ディジタル化の主役 DSP, bit 誌, Vol. 19, No. 14, pp. 83-89 (1989).
10) 持田：DSP の現状と動向, 情報処理, Vol. 30, No. 11, pp. 1291-1299 (1989).

(平成 2 年 7 月 24 日受付)
(平成 2 年 11 月 13 日採録)

向井 信彦（正会員）

昭和 35 年生. 昭和 58 年大阪大学基礎工学部機械工学科卒業. 昭和 60 年同大学院修士課程修了. 同年三菱電機(株)入社. 同社情報電子研究所エンジニアリングシステム開発部所属. 現在グラフィックスの研究・開発に従事. 画像処理, ワークステーションに興味を持つ. SC 24/WG 2 委員.